School of Computation, Information and Technology
Technische Universität München

Übung 11: Pipelining

Einführung in die Rechnerarchitektur

Michael Morandell

School of Computation, Information and Technology
Technische Universität München

13. – 19. Januar 2025

Mitschriften & Infos

Montags:
https://zulip.in.tum.de/#narrow/stream/2668-ERA-Tutorium—Mo-1000-4

Donnerstags:
https://zulip.in.tum.de/#narrow/stream/2657-ERA-Tutorium—Do-1200-2

Website: https://home.in.tum.de/ momi/era/

Keine Garantie für die Richtigkeit der Tutorfolien.

Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

Inhaltsübersicht

- Quiz
- Wiederholung
- Tutorblatt
  - Pipelining Speedup
  - Flushen der Pipeline ohne Hazard Unit (Digital)
  - Pipelining mit Hazard Unit

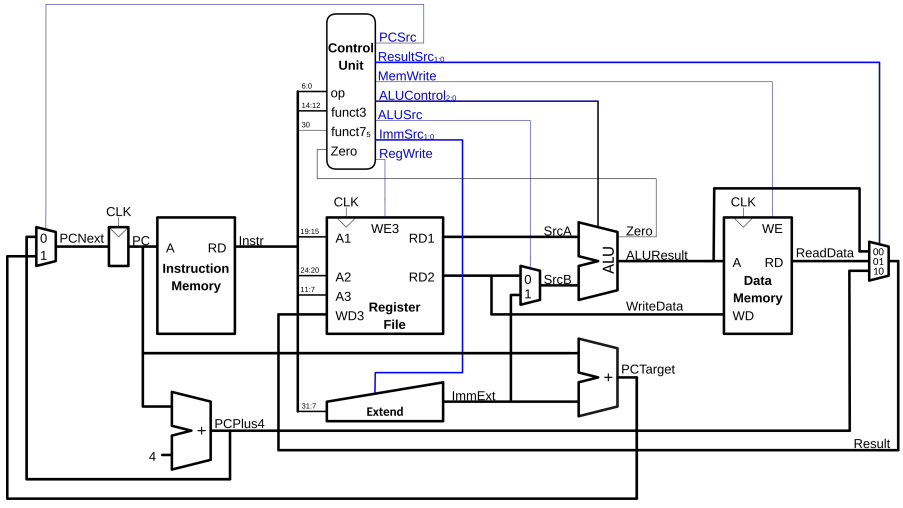
Für die Klausur anmelden! (Frist: 15. Januar)

Prüfungstermine zur Veranstaltung
IN0004 24W 6SWS FA Einführung in die Rechnerarchitektur

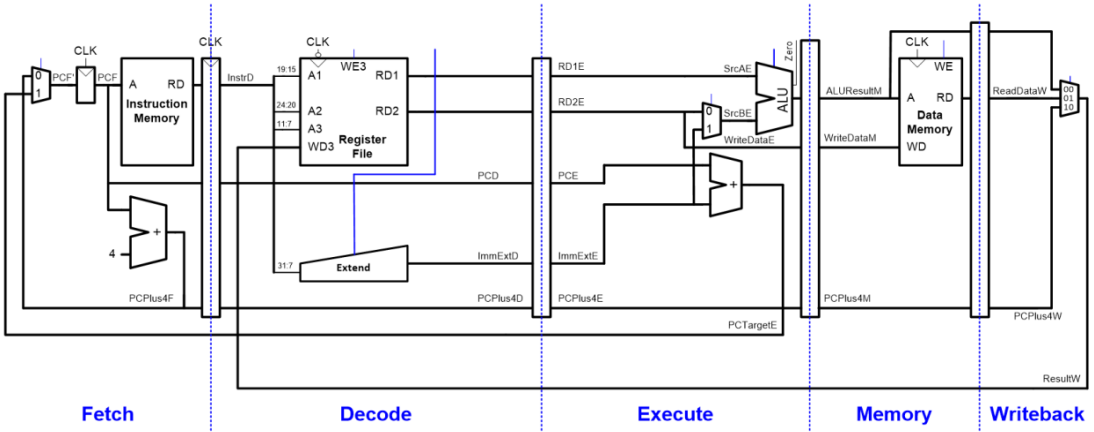
aktuelle(s) LV/Fach

Nr.	Sem.	Art	Titel	Prüfer*in	Datum	Ort	Universität	Anmelde-fenster
Nr. IN0004	Sem. 24W	Art FA	Titel Einführung in die Rechnerarchitektur	Prüfer*in Schulz M (P)	Datum **14.04.2025** 08:00 - 10:00	OrtAudimax im Galileo nur Mo-Di 7-19 Uhr (8120.01.101), MW 0001, Hörsaal (5510.EG.001)	TUMonline	Anmelde-fenster17.03.2025 00:00 31.03.2025 23:59
Nr. IN0004	Sem. 24W	Art FA	Titel Einführung in die Rechnerarchitektur	Prüfer*in Schulz M (P)	Datum **14.02.2025** 08:00 - 10:00	OrtCH 21010, Hans-Fischer-Hörsaal (5401.01.101K), 101, Hörsaal 1, "Interims I" (5620.01.101), 7 weitere(r) Prüfungsort(e)	TUMonline	Anmelde-fenster18.11.2024 00:00 15.01.2025 23:59

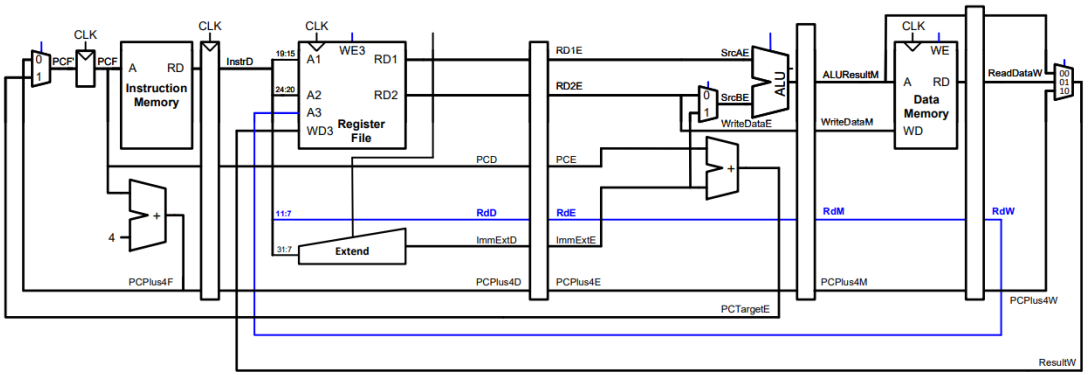
Vom Single Cycle zum Pipeline-Prozessor:



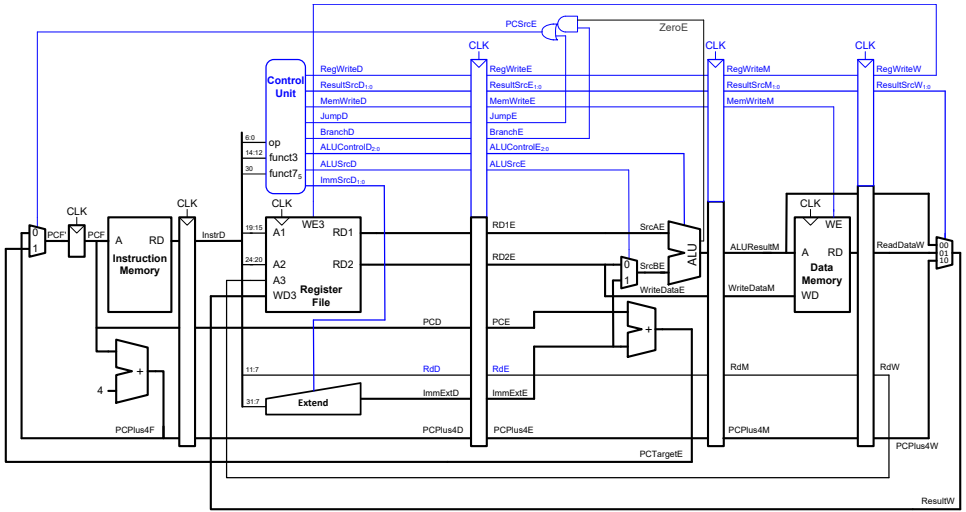
1) Einfügen von 1 FF pro Pipeline-Stufe



2) Datenpfad korriergieren



Fertig ist der Pipelining-Prozessor



Pipelining

- Parallele Verarbeitung von mehreren Instruktionen
- Aufteilung der Instruktionsverarbeitung in 5 Teilschritte: Fetch, Decode, Execute, Memory, Writeback
- Daten- und Kontrollpfad des Prozessors wird aufgeteilt: Register zur Zwischenspeicherung dazwischen
- maximaler Speedup: Anzahl n der Pipelinestufen, Effekt bei großer Zahl an Instruktionen erkennbar

Datenabhängigkeiten

```
s1: lw t0, 0(a1)
s2: lw t1, 0(a0)
s3: add t2, t0, s5
s4: xor a0, t2, a1
s5: lw a0, 0(t1)
s6: sub t2, t3, t0
```

Mit * markierte Abhängigkeiten sind Datenkonflikte

i	F	D	E	M	W
1	s1	-	-	-	-
2	s2	s1	-	-	-
3	s3	s2	s1	-	-
4	s4	s3	s2	s1	-
5	s5	s4	s3	s2	s1
6	s6	s5	s4	s3	s2
7	?	s6	s5	s4	s3
8	?	?	s6	s5	s4

Read after Write (RAW):

i. `s3, s1, t0*`
ii. `s4, s3, t2*`
iii. `s5, s2, t1`
iv. `s6, s1, t0`

Datenabhängigkeiten

```
s1: lw t0, 0(a1)
s2: lw t1, 0(a0)
s3: add t2, t0, s5
s4: xor a0, t2, a1
s5: lw a0, 0(t1)
s6: sub t2, t3, t0
```

Mit * markierte Abhängigkeiten sind Datenkonflikte

i	F	D	E	M	W
1	s1	-	-	-	-
2	s2	s1	-	-	-
3	s3	s2	s1	-	-
4	s4	s3	s2	s1	-
5	s5	s4	s3	s2	s1
6	s6	s5	s4	s3	s2
7	?	s6	s5	s4	s3
8	?	?	s6	s5	s4

Read after Write (RAW):

i. `s3, s1, t0*`

ii. `s4, s3, t2*`

iii. `s5, s2, t1`

iv. `s6, s1, t0`

Write after Read (WAR):

v. `s4, s1, a1`

vi. `s4, s2, a0`

vii. `s5, s2, a0`

viii. `s6, s4, t2`

Datenabhängigkeiten

```
s1: lw t0, 0(a1)
s2: lw t1, 0(a0)
s3: add t2, t0, s5
s4: xor a0, t2, a1
s5: lw a0, 0(t1)
s6: sub t2, t3, t0
```

Mit * markierte Abhängigkeiten sind Datenkonflikte

i	F	D	E	M	W
1	s1	-	-	-	-
2	s2	s1	-	-	-
3	s3	s2	s1	-	-
4	s4	s3	s2	s1	-
5	s5	s4	s3	s2	s1
6	s6	s5	s4	s3	s2
7	?	s6	s5	s4	s3
8	?	?	s6	s5	s4

Write after Read (WAR):

v. `s4, s1, a1`

vi. `s4, s2, a0`

vii. `s5, s2, a0`

viii. `s6, s4, t2`

Write after Write (WAW):

ix. `s5, s4, a0`

x. `s6, s3, t2`

Datenkonflikte

- Daten*abhängigkeiten*: RAW, WAR, WAW
- Pipelinekonflikte: Datenkonflikte (data hazards) und Steuerkonflikte (control hazards)
- Daten*konflikte* können nur bei RAW auftreten (müssen aber nicht): Abhängige Instruktion ist in der Execute-Phase, aber das Ergebnis wurde noch nicht zurückgeschrieben
- Steuerkonflikte treten bei Änderung des Kontrollflusses auf (branches, jumps)

Lösung von Konflikten

Bei **data hazards** müssen mindestens **3 Befehle** zwischen zwei Instruktionen mit RAW-Abhängigkeit stehen:

- NOPs (Stalling)
- Befehlsumordnung (ohne Änderung der Semantik)
- Forwarding: noch nicht zurückgeschriebenes Ergebnis kann von der ALU direkt an den nächsten Befehl gegeben werden, falls dieser das Ergebnis benötigt

Bei **control hazards** müssen mindestens **2 Befehle** zwischen der Sprungentscheidung und möglicherweise falsch geladenen Instruktion stehen.

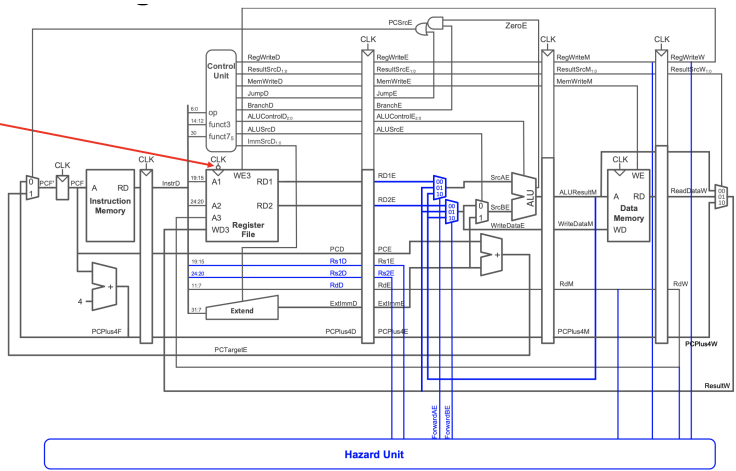
- NOPs (Stalling)
- Branch Prediction (statisch/dynamisch): Falls Vorhersage falsch, müssen geladene Instruktionen entfernt werden

weitere Konzepte: Out-of-Order-Execution, Register Renaming, ...

Forwarding

```
S1 addi s8, s4, 5
S2 sub  s2, s8, s3
S3 or   s9, t6, s8
S4 and  s7, s8, t2
```

- Hazard Unit prüft ob Quellregister des Befehls in Execute mit dem Zielregister in Memory **oder Writeback** übereinstimmt
- Muss für beide Register geprüft werden
- Kein Forwarding von Memory zu Decode nötig



Feedback

https://tinyurl.com/era-tut

Ein Teil der Folien stammt aus dem Foliensatz von Niklas Ladurner. Vielen Dank dafür!